PROxy Based Estimation (PROBE) for Structured Query Language (SQL)

Rob Schoedel, *Microsoft Corporation*

May 2006

Software Engineering Process Management

Technical Note
CMU/SEI-2006-TN-017

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

List of Figures

List of Tables

Acknowledgements

Many people influenced the creation of this relatively short paper. To begin with, the Personal Software Process (PSP) at Microsoft would not have been possible without the strong and ever-present management support of Aidan Waine. After successfully deploying the Team Software Process (TSP) throughout his organization, he skillfully influenced his peers to follow his lead. I'd like to thank the instructors who taught me the PSP and TSP skills: Dan Burton, Julia Mullaney, Bob Musson, and Jim Over. I'd also like to thank several students from my early PSP courses whose names I can no longer remember. Their questions and occasional frustration using SQL in the class assignments led me to look for a better way. I'd also like to thank Eric Woo for trying this technique during a recent PSP course and subsequently letting me publish his data.

Regarding this paper, I wish to thank Watts Humphrey not only for documenting the PROxy Based Estimation (PROBE) technique in the first place, but for taking the time to review my paper and providing comments. Claire Dixon provided wonderful editorial support, masterfully and gently turning mangled sentences into something that others would actually understand!

And finally, this paper wouldn't have come about had it not been for Noopur Davis. I had presented on this topic at an annual TSP conference, and Noopur encouraged me to publish a paper based upon the data. When I actually took her up on it a few months later, she made the publishing aspect very easy – personally reviewing the paper, and then guiding it through all the necessary reviewing and editorial support.

Abstract

This paper presents a method for applying the PROxy Based Estimation (PROBE) technique to Structured Query Language (SQL). Estimating program size is a critical component of successful software project effort estimation and cost estimation. The PROBE technique is a simple estimation method that can be used for estimating program size and effort. To date, PROBE has been used more often to estimate programs written in third-generation programming languages (3GL) such as C, C++, and Java. Its application to IT development has been inhibited by the lack of demonstrated applicability to database work. For data storage, most IT departments have transitioned from file-oriented storage (accessed by traditional 3GL languages) to relational database server software, which uses an implementation of 4GL languages such as SQL to manipulate data. SQL's logic encapsulation properties differ dramatically from those of traditional 3GL languages, so it is not clear to most developers how to effectively apply the PROBE techniques to SQL. The method presented here enables a level of estimation detail similar to the application of PROBE to traditional 3GL languages.

1 Introduction

> It is the mark of an instructed mind to rest satisfied with the degree of precision to which the nature of the subject admits and not to seek exactness when only an approximation of the truth is possible.
> -- *Aristotle, 350 B.C.*

> In the beginning God created Heaven and Earth, Gen. 1, v. 1. Which beginning of time, according to our Chronologie, fell upon the entrance of the night preceding the twenty third day of Octob[er] in the year of the Julian [Period] 710. The year before Christ 4004.
> -- *Archbishop Ussher of Ireland, A.D. 1658*

While historic scholars such as Aristotle and Archbishop Ussher may have grappled with the applicability of estimating precision, many software teams avoid discussing the subject. Estimating is a critical aspect of successful software projects, yet few teams are equipped to produce precise estimates, let alone accurate ones. The Personal Software Process (PSP^{SM}) teaches developers a simple estimation method which can be applied at the time of *conceptual design,*[1] the PROxy Based Estimation (PROBE) technique. The PROBE technique estimates program size by applying historical size data to a conceptual design, and then applying statistical techniques to adjust the estimate based upon past estimating accuracy. By its very nature, the PROBE technique produces precise estimates. The evidence shows that PROBE improves estimation accuracy for developers writing programs in traditional compiled languages such as C++ [Humphrey 02, 05].

However, the comprehensive application of PROBE to Information Technology (IT) development has been inhibited by the lack of demonstrated applicability to database work. During the last 10 years, most IT departments have transitioned their data storage mechanisms from file-oriented storage (accessed by traditional compiled languages) to relational database server software, such as Microsoft SQL Server, IBM DB2, and Oracle Database. To manipulate the data in these servers, each of these products use an implementation of Structured Query Language (SQL). While SQL can be used to create powerful programs, its logic encapsulation properties differ dramatically from those of

[1] Conceptual design is typically performed after the system has been specified functionally, but prior to any detailed technical design. The conceptual design is not intended to be the final design; rather it's a statement from the developer —"if I had this list of components/objects, I could implement the functionality specified."

traditional compiled languages. Thus it is not clear to most developers how to effectively apply the PROBE techniques using SQL.

This paper will present one method of applying the PROBE technique to SQL, at a level of estimation detail similar to the application of PROBE to traditional compiled languages. It is assumed that the reader is familiar with the PSP and with the PROBE estimation method.

2 The PROBE Process

2.1 Introduction and Example

The PROBE technique inherently relies upon a computer science principle called separations of concerns.[2] Over time, a PSP developer will cultivate a categorization of the most frequently used concerns, and collate actual historical size data for those concerns. When making a new estimate, the developer will base his or her estimate on the type of concern being implemented, and the corresponding relative size.

For example, suppose Deborah is a C# developer. She has been tasked to develop a program to read the current location of a train from a database, and then update a display screen based upon the data. After separating the concerns of data access and screen update, and then enumerating the number of methods needed for each class, Deborah concludes that she will need three methods to access the database, and seven methods to update the screen. Consulting her PROBE relative size table for C#, Deborah finds that the average size of a data access method is 8 lines of code, while the average size of an animation method is 19 lines of code. Deborah then uses the PROBE procedure to apply some statistics based upon her past estimation accuracy, and arrives at an estimated program size, including a confidence interval for the estimate.

2.2 Separations of Concern in SQL

The difficulty that development groups have experienced in applying the PROBE technique to SQL is not endemic to PROBE, but rather to SQL. With SQL, the traditional approaches to separation of concerns do not generally apply; the techniques available in popular languages such as classes and methods are largely unavailable in SQL. Thus many SQL developers may conclude that the only way to separate concerns properly and apply PROBE is to create a small stored procedure for each logical concern, which has the undesirable effect of potentially cluttering the database with thousands of stored procedures.

[2] The term *separation of concerns* is credited to Edsger Dijkstra. It is the process of breaking a program into distinct features that overlap in functionality as little as possible. A concern is any piece of interest or focus in a program. Typically, concerns are synonymous with features or behaviors. All programming paradigms aid developers in the process of separation of concerns. For example, object-oriented programming languages such as the Java programming language can separate concerns into classes and methods. Procedural programming languages such as C and Pascal can separate concerns into procedures. For more information, see http://en.wikipedia.org/wiki/Separation_of_concerns [Wikipedia 06].

In actuality, the SQL language is structured such that each statement can be considered a concern. The logic for retrieving data is encapsulated within a SELECT statement, for removing data in a DELETE statement, and so forth. Thus a stored procedure is itself a separation of concern, and each data manipulation statement with the procedure can be considered a separation of concern as well.

2.3 SQL Formatting

The formatting of a SQL statement is important to the success of the PROBE method. For ease of reading and maintenance, it is generally recommended that SQL developers format a single statement across many lines.[3] An example of a T-SQL SELECT statement is shown below. Note that by using such a standard, the statement will grow as more columns are added, or as more tables are joined, or as more filtering conditions are specified. Thus it’s possible for a developer to conceptualize the relative size of a SELECT statement (small, medium, or large) after considering how many fields of data are involved, which tables are involved, and the conditions required for the query.

```
   SELECT C.Name
        , E.NameLast
        , E.NameFirst
        , E.Number
        , ISNULL(I.Description,'NA') AS Description
     FROM tblCompany AS C
     JOIN tblEmployee AS E
       ON C.CompanyID = E.CompanyID
LEFT JOIN tblCoverage AS V
       ON E.EmployeeID = V.EmployeeID
LEFT JOIN tblInsurance AS I
       ON V.InsuranceID = I.InsuranceID
    WHERE C.Name LIKE @Name
      AND V.CreateDate > CONVERT(smalldatetime,
      '01/01/2000')
 ORDER BY C.Name
        , E.NameLast
        , E.NameFirst
        , E.Number
        , ISNULL(I.Description,'NA')
```

Figure 1: Example of a T-SQL SELECT Statement

[3] For Transact-SQL formatting recommendations, see Microsoft SQL Server Professional, December 2004, “T-SQL Coding Standards,” available online http://msdn.microsoft.com/sql/default.aspx?pull=/library/en-us/dnsqlpro04/html/sp04l9.asp. For PL/SQL recommendations, see Purdue University’s Web site, http://www.itap.purdue.edu/ea/data/standards/plsql.cfm.

3 PROBE: The Conceptual Design, and Estimating Parts Size

3.1 Introduction

The completed conceptual design drives the developer’s estimate for designing, coding, and testing a given component. Creating an accurate size estimate relies upon having relevant historical data. The developer creates this historical data by categorizing the types of code produced over time. After applying some statistics, the developer has a table of the types of code written, along with a spread of typical sizes for the code. This table is referred to as the relative size table.

When a developer creates an estimate, he or she lists out all of the parts which will be necessary to create a fully functioning program. The parts are then refined until they match a category in the relative size table.

3.2 Number of Categories

You can have any number of categories in your database, but it is best to keep the number relatively small—just enough to represent the work you do. The Software Engineering Institute recommends starting with a small set of categories, and only making a new category when your data show that you need one. Experience shows that fewer than 10 categories works best. More granularity than that tends to make conceptual design too time consuming and the resulting design over specified. Remember that PROBE aims for broad categories and quick calls on statement types. Further specifying of the design is best left to the detailed design phase.

For SQL, keeping to a small number of categories is fairly simple. There are four major data manipulation commands (SELECT, INSERT, UPDATE, DELETE) plus a handful of other commands that typically span several lines, such as DECLARE.

There are also many commands which typically span only a single line, such as SET statements or control-of-flow language. Should a developer try to count all of the single-line statements as well? With PROBE, I recommend using the 80/20 rule. Focus on those statements which make up 80% of the code, and don’t worry so much about getting the other 20% (which would take a lot more effort to tally). Over time, the statistical methods in the PROBE technique will adjust for the missing 20%. For more on the viability of this approach, see Appendix 1: Study of Stored Procedures.

3.3 Options for Further Consideration

There are a few ways to improve on the 80/20 rule suggested above. If your code counter is sufficiently sophisticated, one option is to make generic PROBE categories for such things as control of flow language (COFL). Certain types of procedures require looping or sophisticated branching logic, while others have none of this. If your LOC counter can tell COFL from regular SQL statements, then you can estimate the number of blocks of COFL statements a particular procedure is going to have, and what size those might be (apart from the SQL statements that may be contained within). However, it's worth noting that the relatively successful results of a limited study discussed later in the paper did not go the extra effort of trying to predict COFL.

3.4 Example Conceptual Design for SQL

Suppose the requirements arrived for a change to an inventory report as follows:

- a new column for the description for Part Type Code
- an update to the way on-hand inventory is calculated
- removal of parts of the report that have not sold in more than one year

The developer would review the requirements for each of these areas, and would spend a few minutes to envision the SQL statements necessary to code the solution.

- A new column for the description for Part Type Code

  Need 1 SELECT (small) from the part_type domain table

- An update to the way on-hand inventory is calculated

  1 UPDATE (medium)

- Remove parts from the report that have not sold in more than one year

  1 INSERT (medium) – into a working table
  1 DELETE (large) – from the actual table

The partially completed size estimation template to accompany this example is shown in Appendix 3: Size Estimating Template Example_

4 Building Historical Data

4.1 Introduction

Applying PROBE to SQL is fairly simple. The more difficult task is obtaining the historical data used to build the relative size table. One way to obtain the historical data would be to simply start having developers make estimates using this approach, and record their findings. Once there is sufficient data, its use as historical data can begin.

However, given that the categorization in PROBE for SQL is objective rather than subjective, we can develop a program to scan the stored procedures and extract statement blocks from these scripts.

4.2 Automating the Gathering of Historical Data

There are a few options for gathering historical data. Conceivably, we could

- build a really smart LOC counter with custom code for each statement type. While this is a viable approach, it would require a significant amount of custom code to handle the multiple variations of the targeted SQL statements.
- build a parser for each script language to recognize statement begin and end. This is a more technically complex, but cleaner approach to recognizing SQL statements. Again a great deal of code would be involved.
- employ a simpler approach, which I've called beginning-of-statement recognition. This method is described in the following section.

4.3 Basic Beginning-of-Statement Recognition

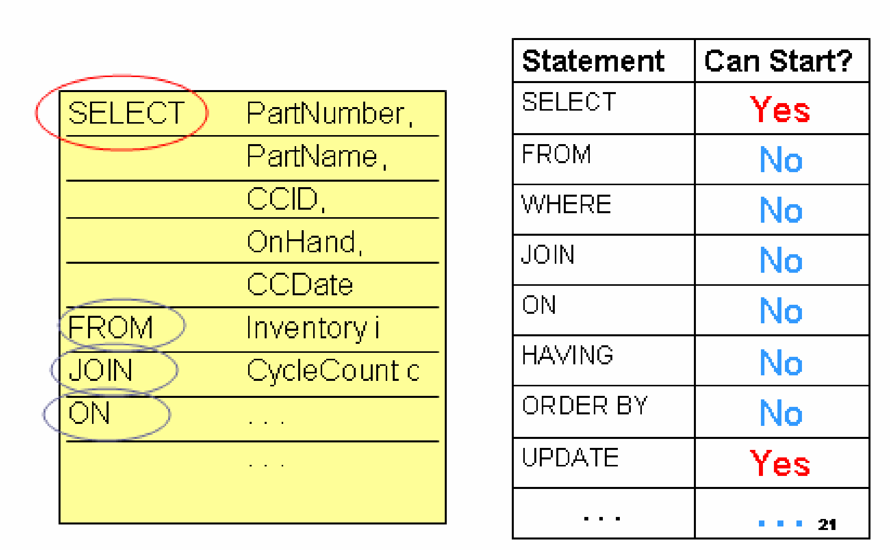
Rather than write a parser, you can employ a simple method to recognize SQL commands within a script, and count the number of lines within each command.

The first step is to consider a SQL stored procedure as a series of lines of code. Consider that only some of the lines will start with a word that is a valid word for beginning a statement. There are a finite number of keywords in the SQL language, and an even smaller number of those keywords represent valid beginning words for a SQL command. So essentially, we can write a line counter in SQL, which looks for statements beginning with certain keywords, such as INSERT, UPDATE, and SELECT. Upon seeing one of these statements we begin counting, and stop counting when we see another valid word which begins another statement. This method is called *basic beginning-of-statement (BOS) recognition*.

The steps in BOS recognition are as follows:

1. Form a list of words that begin a statement (i.e., BOS keywords).
2. Break the script into a list of lines.
3. Iterate through the list until you find a BOS keyword.
4. Begin counting and continue iterating until you find any other BOS keyword.

Table 1: Sample Lists for finding BOS Keywords



```
SELECT    PartNumber,
          PartName,
          CCID,
          OnHand,
          CCDate
FROM      Inventory i
JOIN      CycleCount c
ON        . . .
          . . .
```

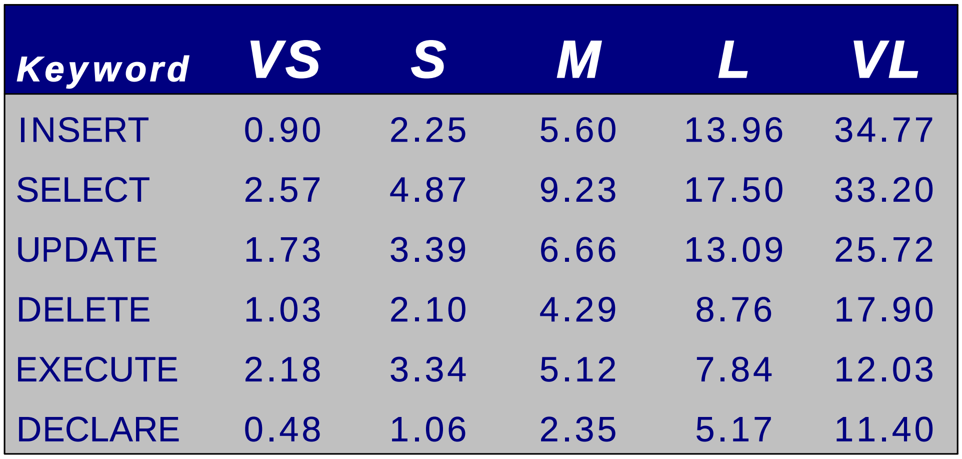
Statement	Can Start?
SELECT	Yes
FROM	No
WHERE	No
JOIN	No
ON	No
HAVING	No
ORDER BY	No
UPDATE	Yes
. . .	. . . 21

The design for building an automated BOS counter is detailed in Appendix 4: Design for BOS Recognition. The BOS recognition method requires adherence to code formatting standards. The best statistics resulting from use of basic BOS recognition will come from code that is similarly formatted, and to agreed-upon standards. As this requirement may be unrealistic for many organizations, a more advanced version of BOS recognition is conceptualized in Appendix 5: Design for Advanced BOS Recognition.

5 Relative Size Table for SQL

The procedure for constructing a data table is detailed in both of the PSP textbooks for engineers. In the study I conducted with one Microsoft business unit (see Appendix 1: Study of Stored Procedures) I used the relative size table process to obtain the following data table.

Table 2: Data Table Obtained Through Relative Size Table Process



Keyword	VS	S	M	L	VL
INSERT	0.90	2.25	5.60	13.96	34.77
SELECT	2.57	4.87	9.23	17.50	33.20
UPDATE	1.73	3.39	6.66	13.09	25.72
DELETE	1.03	2.10	4.29	8.76	17.90
EXECUTE	2.18	3.34	5.12	7.84	12.03
DECLARE	0.48	1.06	2.35	5.17	11.40

This table is suitable for students in PSP who wish to use SQL to complete the construction procedure detailed in the textbooks. This table may also be a good place for Team Software Process team members to start as well. However, because formatting conventions vary, as do database vendor implementation details, I recommend constructing your own table when possible.

6 Areas for Future Study

I have some limited data (see Appendix 2: PSP Student Study) which show good correlation between program size and development time. I'd like to continue to build upon this data, and request that readers of this document who implement PROBE with SQL share their results with me. I can be reached at rob.schoedel@microsoft.com.

One point raised by developers is that the time necessary to develop stored procedures may better correlate to complexity rather than to number of lines of code. For example, a query which involves a join between eight tables would require more care and consideration (and thus time) than a query from a single table which returns 30 columns. Each of these two might have approximately the same number of lines of code. I would encourage further study of this concept. While the LOC method shown here has produced good results in limited usage, it might be true that a multi-factored estimation procedure could produce better estimates for complex code. When piloting an estimation technique, it's important to observe whether developers can use the technique to quickly produce estimates. If in using the technique, the developers spend considerable time in detailed study, then the technique may not be appropriate for conceptual design.

One way to study which elements of a query most contribute to development size and time could be through implementation of the design shown in Appendix 5: Design for Advanced BOS Recognition.

Appendix 1: Study of Stored Procedures

To build a PROBE table for SQL, I analyzed 1314 stored procedures in a transaction-oriented SQL Server database. Because many of the procedures were long enough to exceed one database page, I analyzed a total number of 1817 pages of procedure text.

The PROBE technique is used to predict total program size. However, with this method, we’re only measuring the lines of code found within the chosen SQL statements. There are other types of code which will be added, such as control of flow language, which would not fall into our estimate.

For this approach to be valid, it seems necessary to demonstrate that the number of LOC in the key statements does correlate to the number of overall LOC. For the 1817 entries, the number of LOC in SQL keywords is predictive of the total LOC, with the correlation coefficient $r^2 \approx 0.58$, with high significance ($p=3.8 \times 10^{-308}$).

Table 3: LOC in Keywords Correlated to Overall LOC

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.760389973
R Square	0.57819291
Adjusted R Square	0.57796051
Standard Error	14.83638343
Observations	1817

	Coefficients	*P-value*	*Lower 95%*	*Upper 95%*
Intercept	30.11932849	3.2E-308	28.83913	31.39953
LOC_in_Keywords	0.935617501	0	0.898828	0.972407

Total LOC = 0.935617501 * LOC_in_Keywords + 30.11932849

The regression-fit plot for this data is shown in the diagram below.

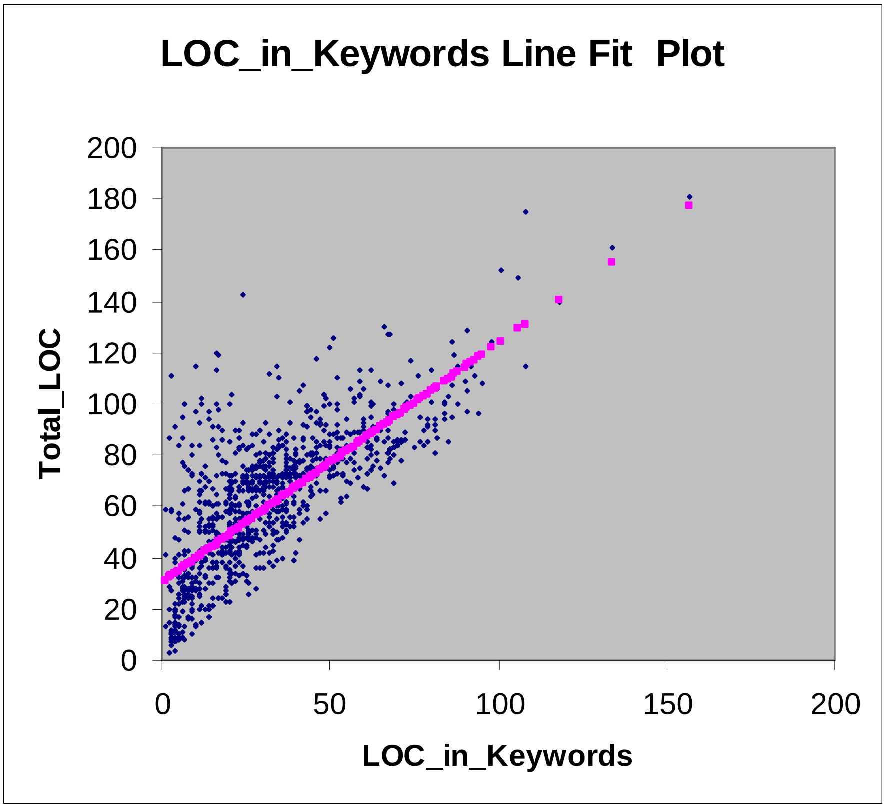


Figure 2:   Regression Fit Line Plot of LOC in Keywords

Appendix 2: PSP Student Study

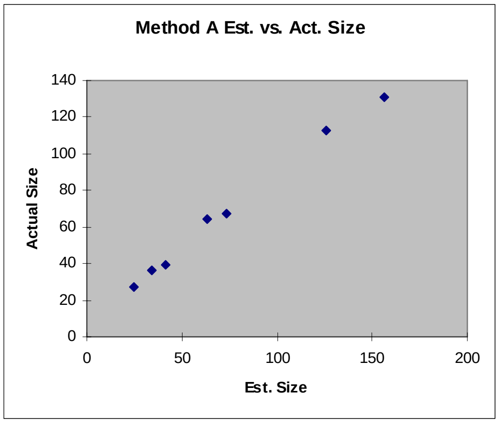
In July 2005, I conducted a PSP course in Beijing, China. One of the developers used the Relative Size Table for SQL to estimate his 10 programs with excellent results for size prediction, and good results for time prediction, illustrated below.

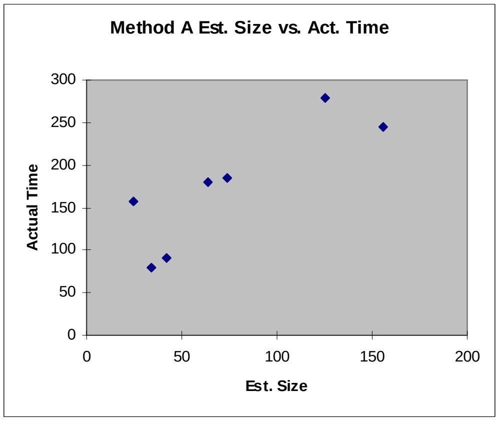
Table 4: PROBE Statistics – Coming in to Program 10

PROBE Method	A		B	
	Size	Time	Size	Time
Estimate	139	327	148	348
R-Squared	0.99	0.76	0.92	0.67
Beta0	5.93	57.30	2.23	66.75
Beta1	0.85	1.73	0.94	1.80
Range (70%)	5	75	17	77
UPI	144	402	166	425
LPI	134	252	131	271
Variance	6.76	1577.56	78.86	1603.16
Std. Deviation	2.60	39.72	8.88	40.04

Table 5: PROBE Statistics – Post Program 10

PROBE Method	A		B	
	Size	Time	Size	Time
R-Squared	0.99	0.72	0.96	0.67
Beta0	8.19	79.80	2.67	86.00
Beta1	0.81	1.27	0.93	1.36
Variance	10.77	1796.36	67.76	1693.36
Std. Deviation	3.28	42.38	8.23	41.15





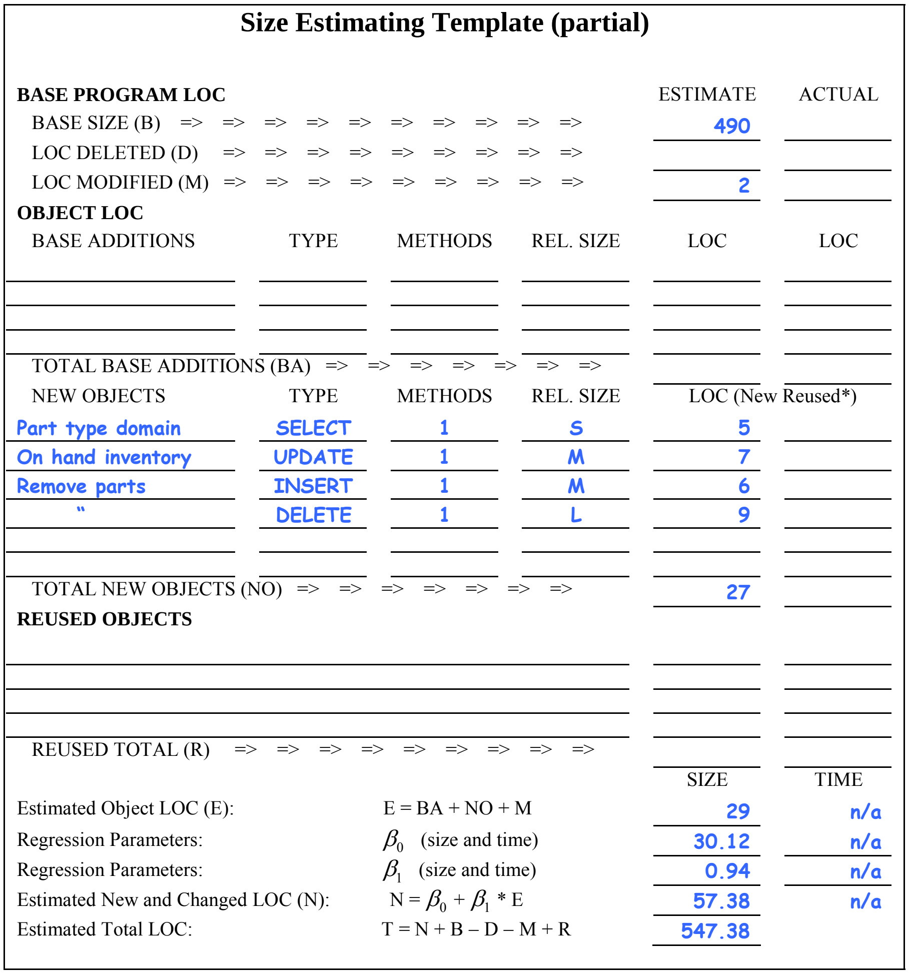
Historical Data

Prog	Est. (N&C)	Actual (N&C)	Actual Min.	Est. (E)
1		20	80	
2	50	30	200	
3	40	38	123.6	
4	19	27	157.4	25
5	38	39	90.55	42
6	115	113	279.3	126
7	34	36	79.9	34
8	68	67	184.2	74
9	59	64	180.8	63
10	139	131	245.1	156

Appendix 3: Size Estimating Template Example

The size estimation template below reflects the exercise data.

Table 6: Example – Size Estimation Template



Size Estimating Template (partial)

BASE PROGRAM LOC				ESTIMATE	ACTUAL
BASE SIZE (B) => => => => => => => => => =>				490	
LOC DELETED (D) => => => => => => => => =>					
LOC MODIFIED (M) => => => => => => => => =>				2	
OBJECT LOC					
BASE ADDITIONS	TYPE	METHODS	REL. SIZE	LOC	LOC
TOTAL BASE ADDITIONS (BA) => => => => => => =>					
NEW OBJECTS	TYPE	METHODS	REL. SIZE	LOC (New Reused*)	
Part type domain	SELECT	1	S	5	
On hand inventory	UPDATE	1	M	7	
Remove parts	INSERT	1	M	6	
"	DELETE	1	L	9	
TOTAL NEW OBJECTS (NO) => => => => => => =>				27	
REUSED OBJECTS					
REUSED TOTAL (R) => => => => => => => => =>					
				SIZE	TIME
Estimated Object LOC (E):	E = BA + NO + M			29	n/a
Regression Parameters:	β_0 (size and time)			30.12	n/a
Regression Parameters:	β_1 (size and time)			0.94	n/a
Estimated New and Changed LOC (N):	$N = \beta_0 + \beta_1 * E$			57.38	n/a
Estimated Total LOC:	T = N + B – D – M + R			547.38	

Appendix 4: Design for BOS Recognition

After a few iterations, I found that the simplest design method is to attribute parentage to the line which contains the beginning-of-statement SQL keyword. The pseudo-code for this design is shown below.

```
LineID=0
For each procedure in the database
   Begin
   For each line in the procedure
      Begin
      LineID++
      If not (line is blank or line is a comment) then
         Begin
         If the first word is a SQL Keyword
            Begin
            Set ParentID = LineID
            Set SQL_Keyword = the first word
            End
         Insert into work table
            Procedure Name,
            LineID,
            ParentID,
            SQL_Keyword,
            Line Text
         End
      End
   End
```

Figure 3:   Attributing Parentage to Line Containing Beginning-of-Statement SQL Keyword

At this point, the work table contains every non-blank line of every stored procedure. Every line refers to its parent line, and also to the SQL keyword it belongs to. From this point, you can query the work table to get summary statistics on each SELECT, INSERT, DELETE, UPDATE, or other SQL keyword in which you’re interested.

One note on this design: INSERT statements that contain a SELECT clause may involve more coding. Consider the example below.

```
INSERT  CoverageSummary
        (CompanyName
        , EmployeeLastName
        , EmployeeFirstName
        , EmployeeNumber
        , InsuranceDescription)
SELECT  C.Name
       , E.NameLast
       , E.NameFirst
       , E.Number
       , ISNULL(I.Description,'NA') AS Description
         FROM tblCompany AS C
         JOIN tblEmployee AS E
           ON C.CompanyID = E.CompanyID
    LEFT JOIN tblCoverage AS V
           ON E.EmployeeID = V.EmployeeID
    LEFT JOIN tblInsurance AS I
           ON V.InsuranceID = I.InsuranceID
        WHERE C.Name LIKE @Name
          AND V.Active = 1
```

Figure 4: Example of INSERT Statement Containing a SELECT Clause

In this example, the design will attribute 6 lines to the INSERT statement and 14 lines to a new SELECT statement. To avoid this and properly attribute 20 lines to the SELECT statement, you can add some additional logic to either the scanning portion of the pseudo-code above, or to the query that is used to summarize the counts.

Appendix 5: Design for Advanced BOS Recognition

My original thought on how to design for counting lines of code in SQL is now affectionately referred to as “advanced BOS recognition,” which simply means that the implementation would have taken too long for my study! The idea is to implement a pseudo-parser, by coding certain elements about the structure of the SQL language into relational entities. If your SQL code base is not formatted uniformly (most are not), this method will produce good results because it is independent of formatting. Instead, it looks for syntax cues regardless of line formatting. The hierarchical command structure can be stored in relational tables. An example of the relationship between statement, clause, and delimiter are shown below, as well as a logical relationship diagram of these entities.

Once a procedure is separated into tables of statements, clauses, and delimiters, you could apply more powerful analysis techniques to study which combination of factors drive the size and development time of stored procedures. Examples of factors revealed in this type of statement would be the number of

- joins in a query
- columns returned
- groupings

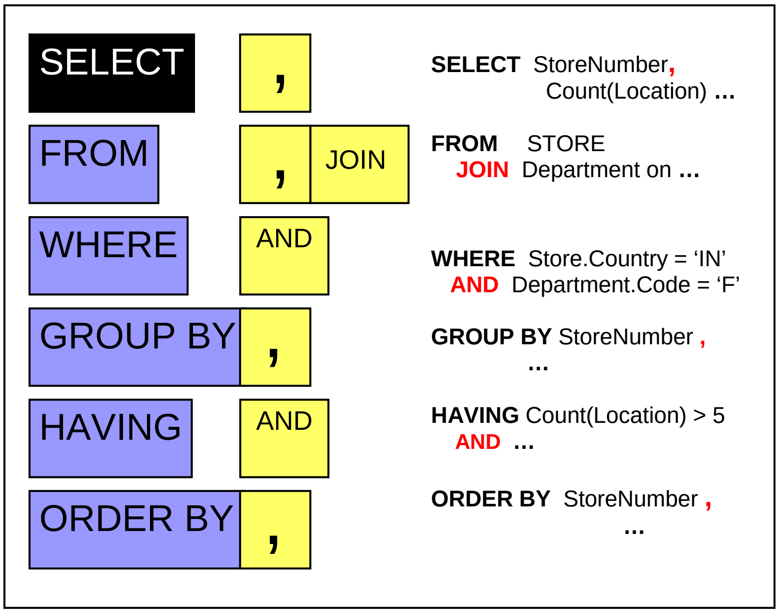


Figure 5: Example of Relationship between Statement, Clause, and Delimiter

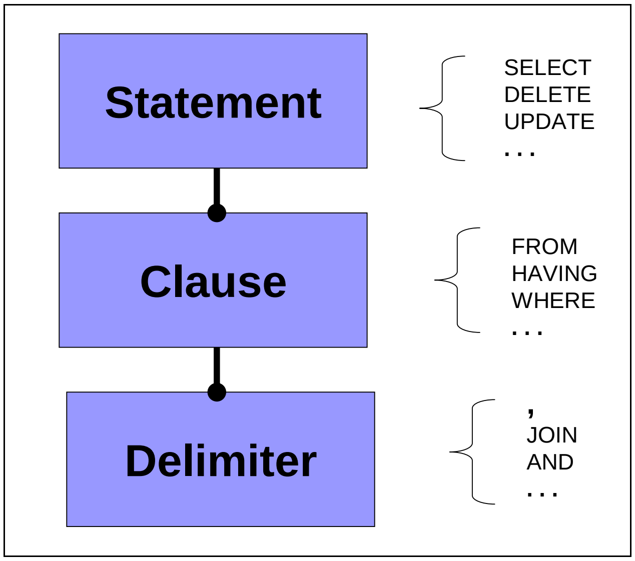


Figure 6: Logical Relationship between Statement, Clause, and Delimiter

References

URLs are valid as of the publication date of this document.

[Humphrey 02] Humphrey, W. S. *Winning with Software*. Boston, MA: Addison-Wesley, 2002.

[Humphrey 05] Humphrey, W. S. *A Self-Improvement Process for Software Engineers*. Boston, MA: Addison-Wesley, 2005.

[Aristotle 350 B.C.] Aristotle, 350 B.C. “Aristotle,” http://en.wikiquote.org/wiki/Aristotle.

[Ussher 1658] Ussher, James. *The Annals of the World iv*. 1658. http://www.nwcreation.net/wiki/index.php?title=The_Annals_of_the_World.

[Wikipedia 06] Wikipedia. “A Separation of Concerns,” http://en.wikipedia.org/wiki/Separation_of_concerns (2006).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY
(Leave Blank)

2. REPORT DATE
May 2006

3. REPORT TYPE AND DATES COVERED
Final

4. TITLE AND SUBTITLE
PROxy Based Estimation (PROBE) for SQL

5. FUNDING NUMBERS
FA8721-05-C-0003

6. AUTHOR(S)
Rob Schoedel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-2006-TN-017

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified/Unlimited, DTIC, NTIS

12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This paper presents a method for applying the PROxy Based Estimation (PROBE) technique to Structured Query Language (SQL). Estimating program size is a critical component of successful software project effort estimation and cost estimation. The PROBE technique is a simple estimation method that can be used for estimating program size and effort. To date, PROBE has been used more often to estimate programs written in third-generation programming languages (3GL) such as C, C++, and Java. Its application to IT development has been inhibited by the lack of demonstrated applicability to database work. For data storage, most IT departments have transitioned from file-oriented storage (accessed by traditional 3GL languages) to relational database server software, which uses an implementation of 4GL languages such as SQL to manipulate data. SQL's logic encapsulation properties differ dramatically from those of traditional 3GL languages, so it is not clear to most developers how to effectively apply the PROBE techniques to SQL. The method presented here enables a level of estimation detail similar to the application of PROBE to traditional 3GL languages.

14. SUBJECT TERMS
estimation, estimate, PROBE statistics, SQL, structured query language

15. NUMBER OF PAGES
35

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT
UL